A Virtual Director Using Hidden Markov Models

Bilal Merabti, Marc Christie, Kadi Bouatouch

Volume xx (200y), Number z, pp. 1–15

A Virtual Director using Hidden Markov Models

B. Merabti[1,2] M. Christie[1,3] K. Bouatouch[1]

[1]IRISA, Rennes, France
[2]Polytechnic Military School, Algiers, Algeria
[3]INRIA Rennes Bretagne Atlantique, France

Abstract

Automatically computing a cinematographic consistent sequence of shots over a set of actions occurring in a 3D world is a complex task which requires not only the computation of appropriate shots (viewpoints) and appropriate transitions between shots (cuts), but the ability to encode and reproduce elements of cinematographic style. Models proposed in the literature, generally based on finite state machine or idiom-based representations, provide limited functionalities to build sequences of shots. These approaches are not designed in mind to easily learn elements of cinematographic style, nor do they allow to perform significant variations in style over the same sequence of actions. In this paper, we propose a model for automated cinematography that can compute significant variations in terms of cinematographic style, with the ability to control the duration of shots and the possibility to add specific constraints to the desired sequence. The model is parameterized in a way that facilitates the application of learning techniques. By using a Hidden Markov Model representation of the editing process, we demonstrate the possibility of easily reproducing elements of style extracted from real movies. Results comparing our model with state-of-the-art first order Markovian representations illustrate these features, and robustness of the learning technique is demonstrated through cross-validation.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Recent possibilities, in rendering increasingly realistic 3D virtual environments in real-time contexts, urge the need for novel techniques to automatically and efficiently convey these contents through the use of appropriate cinematographic techniques (deciding where to place the cameras and how to cut between the cameras). Indeed, there is a clear shift towards a more and more cinematographic experience of 3D environments especially in gaming contexts. More specifically, the reproduction of elements of cinematographic genre and style (war scenes, investigation scenes, etc.) plays a key role in the immersion of users. However most applications rely on a manual preset of camera viewpoints, pre-authored camera paths and triggered cuts between viewpoints. At the cost of a significant manual work, however, the provision of such a set of elements still enables the creation of convincing cinematographic sequences adapted to the locations, and the actions performed by the players.

In attempts to replace this manual endeavor, different techniques have been proposed in the literature to automatically compute cinematographic sequences by relying on film-tree representations [CAH*96, ER07], or evolutions of idiom-based representations [HCS96]. Film-tree representations encode a cinematographic sequence as a set of scenes, further decomposed into shots and into frames. The automatic construction of a movie using a film-tree representation then consists in searching for the best shots and best cuts between frames in shots. While appraised for its generality, such a representation remains limited by the difficulty of parameterizing the search in the film-tree to generate a sequence corresponding to a given style. On the other hand, idiom-based representations require the specification of cinematic idioms (an idiom is a stereotypical way of shooting an action). These idioms are mostly specific, and the mechanisms of transition between idioms are complex and the process therefore lacks generality. Discourse-based approaches [JY05] provide better foundations for the construc-

tion of a narrative discourse by exploiting causal links in events and proposing visual discourse patterns to convey the narrative. Once again, however, the discourse patterns rely on idiom-based representations.

Beyond these limitations in models, a key challenge that has not been addressed by literature is the capacity of such models to learn elements of cinematography from real movies. Questions arise on what is an appropriate model for learning, which type of information is necessary, and how flexible is the model once its parameters have been learned.

In this paper, we propose a general model for the off-line shooting and editing of cinematographic sequences in 3D which learns elements of style from real movies. The proposed representation is able to compute significant variations in terms of cinematographic style, with the ability to control the duration of shots, and the possibility of adding specific constraints to the computed sequence with learned parameters. The proposed model is shown to be effective in reproducing elements of style from real sequences. The approach is founded on a Hidden Markov Model representation of the editing process, where the states represent the shots and the observations are the events that the shots are portraying.

Our contributions are threefold:

- a better characterization of events in dialog-based sequences by shifting towards a discursive representation of events (e.g. moral positions of characters) rather than plain events adopted by a large number of techniques (e.g. a character is talking) [HCS96, ER07, LCL*10, LCCR11]
- a general representation of the editing process that enables learning and reproducing elements of style from real sequences, with variations in the pacing (rhythm of shots) and possibility to add constraints.
- a representation which is independent of the geometry and relies on a geometric solver to place cameras automatically.

2. Related work

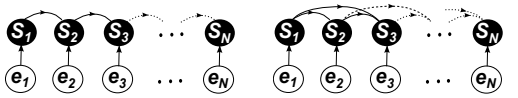
A number of approaches have been proposed in the literature to partially or fully automate the computation of viewpoints and edits. The seminal work of He et al. [HCS96] proposes to encode the directorial process (placing the cameras and selecting the cuts between the cameras) as a finite state machine (FSM). A state in the FSM represents a canonical shot (*e.g.* apex shot, over-the-shoulder shot or panoramic), while a transition between states represents a cut. The transitions are either triggered by scene events (*e.g.* when the distance between two characters reaches a threshold) or temporal events (duration of the current shot). The FSM representation is then used in a real-time context to compute a cinematographic sequence: as the 3D scene evolves, the FSM places the camera corresponding to the shot expressed in the current state, and performs the cut when events occur. By relying on Arijon's cookbook [Ari76] of camera setup, the authors encode different film idioms (stereotypical ways of shooting character configurations and actions) as different finite state machines, organized in a hierarchical representation that enables transitions between FSMs. This hierarchical representation can be viewed as a mean to encode some aspects of film style (choice in shots, rhythm of cuts). However, the nature of the triggers on the transitions together with the complexity of associating idioms with all possible character configurations restricts the applicability of the technique to well-known scenarios. Furthermore, it is necessary to specify different FSMs to encode different film styles, and set up mechanisms of transitions between the FSMs.

In an approach that focuses on portraying the motion of characters in a 3D scene, Assa et al. [AWCO10] estimate the quality of a collection of viewpoints by measuring the correlation between the spatial motions of characters' articulations and the on-screen projected motions of the same articulations. The selected viewpoint is the one displaying the best correlation. A viewpoint erosion cost is added to force cuts between viewpoints (the cost is a function of the duration of the shot). Extra costs account for violation of continuity editing rules. The method however requires to modify the costs and correlation metrics to propose different ways of shooting the same sequence.

In order to improve the cinematographic quality of such systems, Jhala et al. [JY05] proposed a bipartite model by establishing a more principled link between the story (defined as a collection of events) and the discourse (defined as means to convey this collection of events). The system has the strength of reasoning over story scenarios, and the proposed discourse planner is based on individual events, at the level of the scene. Patterns of shots must however be defined in advance for different scenes and events taking place (such as Show-Bank-Robbery), thereby restricting the applicability of the system to different stories, or at least requiring to create the appropriate shot patterns.

Other approaches rely on the film-tree representation (initially proposed by Christianson et al. [CAH*96]) that represents a movie as a hierarchical structure of sequences decomposed into scenes, candidate idioms and candidate frames. Computing the best sequence of frames consists in searching for the best frames, the best idioms and the best transitions between idioms by exploring and evaluating all the possibilities in the film tree, similar to a combinatorial branching search. By far the most general representation in cinematographic editing, inspiring many approaches [LCCR11, ER07], film-trees actually require the provision of many parameters and weights to guide the search process, therefore hampering their use in practice.

A complete cinematographic process has been proposed by Elson and Reidl [ER07] that deals simultaneously with tasks of blocking (placing the characters and the scene), shooting and editing. The authors rely on a combination



a) first order dependency shots. b) second order dependency shots.

Figure 1: Shot dependency.

of search and dynamic programming to identify the best shots and best blockings, by processing a library of canonical shots associated with canonical blockings. The search phases explores the best locations where to stage the scene, while the dynamic programming phases searches for the best sequence of shots to convey the beats (a beat is a sequence of actions, representing a unit in the narrative). While the approach enables variations in the style (through directorial heuristics), the control of these variations is performed through a manually defined system of weights.

Inspired by these previous contributions, one can actually generalize them in a simplified model and view a film as a collection of shots, each shot portraying an event in the story. Such a generalization accounts for both idiom-based approaches (that constrains to use a subsequence of shots associated with an event), and for film-tree representations (however not considering the level of frames in shots which makes the search process computationally hard).

More formally, let X be the set of all possible shot descriptions in a cinematographic language and let E be the set of all event types existing in a given story. The task of any automated editing process as proposed in [HCS96] or [ER07] is to associate the best sequence of shots $\dot{s} = s_1, s_2, \cdots, s_N$, with $s_i \in X$ within all possible shot sequences $\hat{s}$ with a given sequence of events $\hat{e} = e_1, e_2, \cdots, e_N$, with $e_i \in E$ (see equation 1).

$$\dot{s} = \arg\max_{\hat{s}}(p(\hat{s}|\hat{e})) \tag{1}$$

Many contributions rely on this representation and consider the editing process as Markovian (i.e. a first order Markov chain).

Considering that each shot s_i depends on one single event e_i and one previous shot s_{i-1} (as shown in Figure 1.a), we can therefore represent the joint probability as:

$$p(\hat{s}|\hat{e}) = p(s_1|e_1)\prod_{i=2}^{N} p(s_i|s_{i-1}, e_i) \tag{2}$$

When using such a model to predict the next shot in a sequence, the distribution of predictions will depend only on the the value of the corresponding event and the value of the immediately preceding shot and will be independent of all earlier shots.

A number of contributions actually rely on this hypothesis (see [ER07,LCL*10,LCCR11]). Authors [LCRB11] proposed this probabilistic representation where states encode shots and transitions are activated by events. Transitions between shots can therefore be represented with a transition matrix in which values are defined using cinematographic rules and preferences in transitions, thereby defining such a first order Markov model. If a shot s takes values from a set $X = \{x_1, x_2, x_3\}$, the transition matrix can be represented as a Markov automaton (see Figure2). And a sequence $\hat{s}$ simply represents a feasible path in the automaton.

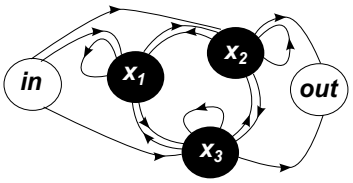


Figure 2: Finite state Markov automaton Illustrating transitions between three shots x_1, x_2 and x_3

However, in many cinematographic sequences, the choice of a shot not only depends on the previous shots but on a number of preceding shots, making the first order Markovian hypothesis very restrictive. One way to account for preceding shots is to switch to higher-order Markov chains. With a second-order Markov chain, as represented in Figure 1.b, the joint distribution is expressed by:

$$p(s_1, s_2, \cdots, s_N) = p(s_1)p(s_2|s_1)\prod_{i=2}^{N} p(s_i|s_{i-2}, s_{i-1}) \tag{3}$$

The joint probability is therefore expressed as:

$$p(\hat{s}|\hat{e}) = p(s_1|e_1)p(s_2|s_1, e_1)\prod_{i=2}^{N} p(s_i|s_{i-2}, s_{i-1}, e_i) \tag{4}$$

Each shot is now influenced by the two previous shots, and we can similarly consider extensions to an M^{th}-order Markov chain in which the conditional distribution for a particular variable depends on the previous M variables.

However, in order to use such a general model for editing, all the parameters need to be instantiated, i.e. associate the right probabilities with the parameters. Let us suppose that the shots are discrete variables having K different states. Then the conditional distribution $p(s_i|s_{i-1})$ in a first-order Markov chain will be specified by a set of $K-1$ parameters for each of the K states of s_{i-1}, giving a total of $K(K-1)$ parameters. Now in an M^{th}-order Markov chain, the joint distribution in equation 3 should be expressed as a product of all the $p(s_i|s_{i-M}, \cdots, s_{i-1})$ terms. If the variables are discrete, and if the conditional distributions are represented by general conditional probability tables, then the number of parameters in such a model will be $K^M(K-1)$. Because this number grows exponentially with M, such an approach

is impractical for larger values of M. For example, in practice for $K = 30$ (i.e. 30 different types of shots are considered), a third-order Markov chain requires setting the values of 8×10^5 parameters.

Furthermore, following the idea of trying to learn elements of shot placement and cuts from real movies, a very significant amount of data would be necessary. In the following, we will show how to address both issues: (i) reducing the number of parameters in an expressive computational model of editing, and (ii) proposing means to learn these parameters from existing movies.

3. Overview

The main objective of our work is to propose an expressive editing model that has the ability to reproduce a given cinematographic style. The first question that arises is how to characterize style. To avoid addressing this vast question, we reduce cinematographic style, in this paper, to the choice of shooting angles wrt. events (eg. medium over the shoulder shot), and possible transitions between these shooting angles. The process uses as input a script describing events occurring in a 3D environment, and as outputs an optimal cinematographic edit of the sequence in which shots and cuts are automatically computed given a learnt style. Learning cinematographic style consists in extracting cinematographic elements such as shot description (which type of shot, which actors are composed in the shot and which ones are in focus), shot transitions (towards which shots the cuts are performed), and relations between shot descriptions and events (see section 5).

3.1. An expressive editing model

A first issue is to address the dimensionality of the problem, i.e. to propose a model with a reduced number of parameters that require learning. Looking back at the first-order Markov model with the joint probabilities (see Figure 1), we need to set as many parameters as those of a simple second-order Markov model. Learning every

$$p(s_i = x_u | s_{i-1} = x_v, e_i)$$

is as complex as learning

$$p(s_i = x_u | s_{i-1} = x_v, s_{i-2} = x_w)$$

Nonetheless, the joint probability can be developed using the Kolmogorov definition:

$$p(s_i|s_{i-1}, e_i) = \frac{p(s_i|s_{i-1}) \times p(s_i|e_i)}{p(s_i)} \quad (5)$$

The use of equation 5 then allows to reduce the number of parameters to learn, compared to the joint probability model. Nevertheless, learning the probability of a shot $p(s_i)$ is problematic since it represents the probability of a shot to appear in a sequence and in some cases can be null. And as it appears as a denominator in Equation 5, it strongly influences the overall probability.

This probability is supposed constant in other contributions such as those based on Finite State Machine representations [CAH*96], i.e. $p(s_i) = p(s_j) \forall i, j$, which is contradicted by empirical evidence in cinematography.

Until now, the classical way of viewing an editing process is by considering there are several ways to shoot each event. Therefore, shooting an individual event e_i means finding the most likely shot s_i that maximizes the probability $p(s_i|e_i)$.

However, one can actually revert the problem by considering that we know the probability of an event given a shot (see Figure 3). And in such a case, given this sequence of events, we are actually looking for the best sequence of shots leading to these events. By considering that shots are states, and that events are observations (an observation being a probabilistic function of a state), our model looks like a fairly known model in the literature: Hidden Markov Model.

In such a case, the parameters of the model are (i) the probability of the first shot, (ii) the probability of an observation (event) knowing a shot, referred to as the emission matrix and (iii) the probability of a shot knowing the previous one, referred to as the transition matrix.

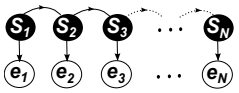


Figure 3: Hidden Markov Model representation of a cinematographic sequence.

3.2. Hidden Markov Models: quick reference and notations

A hidden Markov model (HMM) is a statistical model in which the system being modeled is assumed to be a Markov process with unknown parameters. HMMs are widely used in pattern recognition, artificial intelligence or automatic natural language processing [BLBA08, Rab89].

Formally, An HMM is a Markov model composed of two types of variables: hidden and observable. Each hidden random variable (shot) $s \in \hat{s}$ takes values from a set $X = \{x_i\}_{i=1}^{|X|}$ called state alphabet. An observed variable (event) $e \in \{e_i\}_{i=1}^{T}$ is drawn from an output alphabet denoted in this paper $O = \{o_i\}_{i=1}^{|O|}$.

HMM representation assumes two *Markov assumptions*:

1. Limited Horizon assumption: that means that for the probability, that a random variable to be in a given state at time t, depends only on the state of the variable at time $t-1$.

$$p(s_t|s_{t-1}, \ldots, s_1) = p(s_t|s_{t-1})$$

2. Stationary process assumption: which means that the transition (resp. observation) are time invariant.

$$p(s_t|s_{t-1},t) = p(s_t|s_{t-1})$$

$$p(e_t|s_t,t) = p(s|s_t)$$

Therefore, an HMM can be parametrized by a tuple $\{X,O,\Pi,A,B\}$ where:

1. A: a square matrix with length $|X|$ which denotes the transition conditional probability tables (CPT) for the HMM. Each element a_{x_i,x_j} (denoted simply a_{ij}) represents the probability that variable s_t be equal to x_j when $s_{t-1} = x_i$.

$$a_{x_i,x_j} = a_{ij} = p(s_t = x_j|s_{t-1} = x_i).$$

2. B: a $|X|\times\ |O|$ sized matrix which denotes the emission CPT.

$$b_{x_i,o_j} = b_{ij} = p(e_t = o_j|s_t = x_i)$$

3. $\Pi = (\pi_{x_k})_{k=1}^{|X|}$ a vector representing the PDF (probability distribution function) for the first hidden variable s_1. π_{x_i} (denoted π_i) is the probability that s_1 is in state x_i.

$$\pi_{x_i} = \pi_i = p(s_1 = x_i)$$

Note that, each line in matrices A and B, and vector Π, describes the PDF of a discrete random variable. Therefore:

$$\sum_{i=1}^{k}\pi_i = 1, \qquad \forall i, \sum_j a_{ij} = 1, \qquad \forall i, \sum_j b_{ij} = 1$$

There are three typical examples of problems that can be resolved with an HMM: evaluating, decoding and learning [Rab89]. The decoding problem consists, given an HMM, in finding out the most probable sequence of hidden states (shots in our case) which generates a given sequence of observations (events). This problem is, in general, solved using a dynamic algorithm of quadratic complexity well known as the *Viterbi algorithm* [Vit67]. The learning problem, which consists in building and parameterizing an HMM, is often addressed using the iterative "Baum-Welch" algorithm [BPSW70].

3.3. Overall process

The overall process is organized in two stages (see Figure 4 and Algorithms 1 and 2). The first stage is the learning stage that requires (i) manually annotating sequences from movies, and (ii) learning the model's parameters (see Algorithm 1). The annotation comprises the description of the events occurring in the shots following a narrative model we devised (see Section 4), and the description of the shots (characters involved, entering or leaving the frame, angle, type of shot, main character and focus). The learning process then consists in setting the values of the HMM parameters related to the transition and emission matrices.

Algorithm 1 Learn(MovieSequences M , Scripts T) :
e : Event Sequence
s : Shot Sequence
h : HMM

```
1: h.initialize() ;
2: for all (m,t) in (M,T) do
3:    e = ScriptAnnotation(t);
4:    s = ShotAnnotation(m);
5:    if s.exists() then
6:       h.updateFrequencies(e, s);
7:    else
8:       h.Baum-Welch-iteration(e);
9:    end if
10: end for
11: return h ;
```

The second stage consists in applying the learned style, in addition to extra constraints, to an annotated script. First, the *Director* encoded as an HMM (see section 4) relies on this annotated script to produce the optimal sequence of shot descriptions. Second, this sequence of shot descriptions is exploited by an automated *Cinematographer* relying on an efficient representation [LC12] to position the camera and set its parameters in a 3D environment (see Section 6).

Algorithm 2 Reproduce(HMM h, Script t, Constraints c)
e : Event Sequence
s : Shot Sequence
cs : Camera configuration sequence
m : Rendered Movie sequence

```
1: e = ScriptAnnotation(t);
2: s = Director(h, e, c) ;
3: cs = Cinematographer(s) ;
4: m = Renderer(cs);
5: return m ;
```

4. Modeling

Before describing the learning process, we propose to detail the components of our model, that is how shots and events are represented.

4.1. Annotated Script (Observations)

The formalization of the input script in our model requires the definition of two categories of events: real events and meta-events (see Figure 5). Real events specify the communicative value of dialogs in the script and we propose to classify these dialogs into three categories: symbolic, moral and narrative. In comparison with most approaches that consider a plain annotation of dialog (for example, a "character speaks" such as in [HCS96, ER07, LCL*10, GRLC14]) we

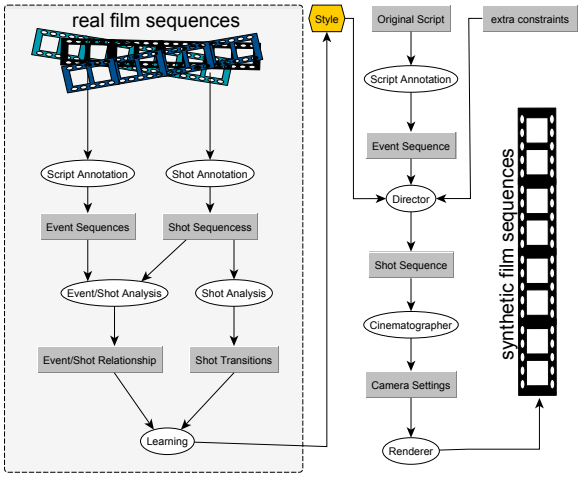


Figure 4: Overall process. On the left, cinematographic style is learned from sequences of annotated shots. On the right, the learned cinematographic style is applied to generate a sequence of shots for a new script, given extra constraints such as targeted duration of shots, or constrained shots.

```
Jack: Gentlemen, what do keys do?
Turkish Pirate: Keys unlock things?
Gibbs: And whatever this key unlocks, inside there's something
     valuable. So we're setting out to find whatever this key
     unlocks.
Jack: No. If we don't have the key, we can't open whatever we
     don't have that it unlocks. So what purpose would be
     served in finding whatever need be unlocked, which we
     don't have, without first having found the key what
     unlocks it?
Gibbs: So we're going after this key.
Jack: You're not making any sense at all.
Jack: Any more questions?
```

a. Original script scene

```
\meta_event{new actor (Jack)}
Jack to All:" \mor{Gentlemen,} \que{what do keys do?}"
\meta_event{new secondary actor(TP)}
Turkish Pirate to Jack: " \que{Keys unlock things?}"
\meta_event{TP replaced by secondary actor(Gibbs)}
Gibbs to Jack: "\nar{And whatever this key unlocks, inside
     there's something valuable.} \nar{So we're setting out to
     find whatever this key unlocks.}"
Jack to Gibbs: " \mpl{No.} \mgl {If we don't have the key, we
     can't open whatever we don't have that it unlocks.}
     \mpl{So what purpose would be served in finding whatever
     need be unlocked,} \nar{which we don't have,} \mpl{without
     first having found the key what unlocks it?}"
Gibbs to Jack: "\nar{So we're going after this key.}"
Jack to Gibbs: "\mpl{You're not making any sense at all.}"
Jack to All: "\que{Any more questions?}"
```

b. Annotated script scene

Figure 5: Original and annotated script

provide a classification of the communicative values of utterances in dialogs. Our approach – that is inspired by the way filmmakers construct dialogs [Bou13] – relies on a fine-grained representation of utterances: symbolic, moral and narrative communicative values. This representation features a stronger relation with the type of shot used to portray a given communicative value.

Meta-events describe changes of actors between real events such as when an important actor in a scene is replaced by another one of less importance. Meta-events are modeled using discrete variables which provide information about the actors entering and leaving the stage such as:

HIGH_HIGH : The actor of the highest importance exits the stage and is replaced by another with a higher importance than the remaining one;

HIGH_LOW : The actor of the highest importance exits the stage and is replaced by another with a lower importance than the remaining one ;

LOW_LOW : The actor of the lower importance exits the stage and is replaced by another with a lower importance than the remaining one ;

LOW_HIGH : The actor of the lower importance exits the stage and is replaced by another with a higher importance than the remaining one ;

BOTH : Both actors are entering the stage and were not participating in the previous one.

This information is inserted into the event list as a meta-event before the real event occurs in the script. These meta-events describe the role of an actor expressed as (i) his importance with respect to other actors in the current event (a solution also considered by [GRLC14]), and (ii) his presence or not in the related events. This allows the modelling of sequences with multiple actors entering and leaving the stage, an aspect only partially addressed by previous work (see Kardan etal. [KC08] for an approach based on hierarchical lines of actions).

4.2. Shot specification (Hidden States)

The design of shots for cinematography consists in considering a number of visual features such as *shot type*, *visual composition*, *focus*, lighting and color [War03]. In our work, we only consider the three first features, that are the ones we annotate in the real movies. These features are also the minimal set for an automated cinematography system to place cameras [LC12], *i.e.* to determine camera position, orientation, zoom and depth of field from such features.

In our representation, a shot s_n is therefore characterized by a shot type given by a value among of set of 4 possible values for one actor shots: fullBody, medium, close, extCloseUp, and among a set of 8 possible values for two actor shots: fullBody2Shots, Medium2Shots, CloseUp2Shots, FullBodyOverShoulder, medOverShoulder, closeOverShoulder, symbolicShot, overallShot. A shot also encodes the importance of the character it shoots (higher or lower importance).

Now, in order to determine which actors are composed in the shot, a second type of hidden state is introduced: the

prep-shot. A prep-shot is a state that helps to prepare a subsequence of shot states by precisely defining who is the actor of higher, respectively lower, importance in the shot. Prep-shot states are associated with meta-events observations in the script. Meta-events define changes between the actors as well as changes in their relative importance, and prep-shot states reflect these changes in the shots.

4.3. Matrix representation

We propose to represent an instance of an HMM (*i.e.* a cinematographic style) as two matrices (see Figure 6). The left and right matrices respectively represent the transition and the emission matrices of the HMM. The transition matrix is a square matrix (figure 6.left) where each element (i, j) provides the probability of transition from a shot x_i to a shot x_j: $p(s_n = x_i|s_{n-1} = x_j)$. The highlighted bottom-right square in the transition matrix represents transitions between shots, while other areas represent the transitions including prep-shots.

Regarding the emission matrix (the rightmost image), each couple (i,k) holds the probability of an event o_k given the shot x_i: $p(e_n = o_k|s_n = x_i)$. The highlighted bottom-right square in the emission matrix represents probabilities of real events regarding shots. While the top-left framed area represents the relationships between meta-events and prep-shots. Probabilities in other areas are set to zero, meaning that there is no relation between meta-events and shots or between real events and prep-shots.

Transition Matrix: each element (i, j) represents the transition probability from state x_i to x_j. Each shot x is of type $F(act)$, where F represents a framing as specified in section 4.2 (*e.g.* FullBody, CloseUp2shots, MedOverShoulder), and *act* the actor in focus. *act* is denoted H if it is of higher importance and L if it is of lower importance. When F starts with "p_", x is a prep-shot rather than a shot. Below, the list of the considered shots (and prep-shots) illustrated by figure 6:
x_1 = p_init(), x_2 = p_set(2actors), x_3 = p_set(1actor), x_4 = p_update(2act), x_5 = p_update(Xact), x_6 = p_update(Yact), x_7 = p_update(Up), x_8 = p_update(Down), x_9 = p_invert(), x_{10} = p_update(Symb), x_{11} = fullBody(X), x_{12} = medium(X), x_{13} = close(X), x_{14} = extCloseUp(X), x_{15} = fullBody(Y), x_{16} = medium(Y), x_{17} = close(Y), x_{18} = extCloseUp(Y), x_{19} = fullBody2Shots(X), x_{20} = Medium2Shots(X), x_{21} = CloseUp2Shots(X), x_{22} = FullBodyOverShoulder(X), x_{23} = medOverShoulder(X), x_{24} = closeOverShoulder(X), x_{25} = fullBody2Shots(Y), x_{26} = medium2Shots(Y), x_{27} = close2Shots(Y) x_{28} = fullBodyOverShoulder(Y), x_{29} = medOverShoulder(Y), x_{30} = closeOverShoulder(Y), x_{31} = symbShot(A), x_{32} = overAllShot(), x_{33} = p_final().

Emission Matrix: each element (i, j) represents $p(y_j|x_i)$. y_i is an event and x_i is a state. Each event y is of type $T(act, dest)$, where T is the action of the event, *act* is the actor making the action, and *dest* is one or more actors to whom the action is addressed. When T starts with "m_", x is a meta-event rather than an event. Below, the list of the considered events (and meta-events) illustrated by figure 6:
y_1 = m_update(X,Y), y_2 = m_updateUp(X,Y), y_3 = m_updateDown(X,Y), y_4 = m_update(X), y_5 = m_update(Y), y_6 = m_update(S), y_7 = neutral, y_8 = symbolic(S), y_9 = symbolic(X,Y), y_{10} = symbolic(Y,X), y_{11} = symbolic(X,All), y_{12} = symbolic(Y,All), y_{13} = narration(X,Y), y_{14} = moralGle(X,Y), y_{15} = moralPle(X,Y), y_{16} = moralOrder(X,Y), y_{17} = question(X,Y), y_{18} = narration(X,All), y_{19} = moralGle(X,All), y_{20} = moralPle(X,All), y_{21} = moralOrder(X,All), y_{22} = question(X,All), y_{23} = narration(Y,X), y_{24} = moralGle(Y,X), y_{25} = moralPle(Y,X), y_{26} = moralOrder(Y,X), y_{27} = question(Y,X), y_{28} = narration(Y,All), y_{29} = moralGle(Y,All), y_{30} = moralPle(Y,All), y_{31} = moralOrder(Y,All), y_{32} = question(Y,All), y_{32}-y_{33}-y_{34}-y_{35}-y_{36}-y_{37} same as y_{11}-y_{18}-y_{19}-y_{20}-y_{21}-y_{22} but the interlocutors here are not yet introduced, y_{38} = unknown() .

5. Learning

HMM learning generally starts with some a priori knowledge about the values of the model $\mathbb{\theta} = \{\mathcal{X}, \mathcal{O}, \Pi, \mathcal{A}, \mathcal{B}\}$. This initial knowledge is presented as a prior probability distribution $p(\mathbb{\theta})$ over the model parameters. The model parameters are then updated using the learning dataset D to obtain the posterior probability distribution $p(\mathbb{\theta}|D)$. Then θ, the learned instance of the model $\mathbb{\theta}$ – that represents a specific assignment of parameters of the model – is computed as:

$$\theta = \arg\max_{\mathbb{\theta}} p(\mathbb{\theta}|D) \tag{6}$$

Recall that, applying Bayes rule we have:

$$p(\mathbb{\theta}|D) = \frac{p(D|\mathbb{\theta})p(\mathbb{\theta})}{p(D)}.$$

Therefore, if the prior over the parameters $\mathbb{\theta}$ is non informative (*e.g.* uniform distribution), the probability $p(\mathbb{\theta}|D)$ will display a peak around the maximum of the likelihood function $p(D|\mathbb{\theta})$. And therefore, θ in equation 6 is estimated as $\dot{\theta}$ by maximizing the likelihood, or the log likelihood as shown in equation 7:

$$\dot{\theta} = \arg\max_{\mathbb{\theta}} \log p(D|\mathbb{\theta}) \tag{7}$$

Note that, in our application, the prior on $\mathbb{\theta}$ is a uniform distribution over all its components: $\{\mathcal{X}, \mathcal{O}, \Pi, \mathcal{A}, \mathcal{B}\}$. The labelled sets $\mathcal{X}$ and $\mathcal{O}$ are assumed to be known (see section 4): $\mathcal{X} = X$ and $\mathcal{O} = O$ (X and O are the realizations of the random variables $\mathcal{X}$ and $\mathcal{O}$ respectively as defined in section 4.3). Therefore equation 7 becomes:

$$\{A, B, \Pi\} = \arg\max_{\mathcal{A},\mathcal{B},\Pi} p(D|\mathcal{A}, \mathcal{B}, \Pi) \tag{8}$$

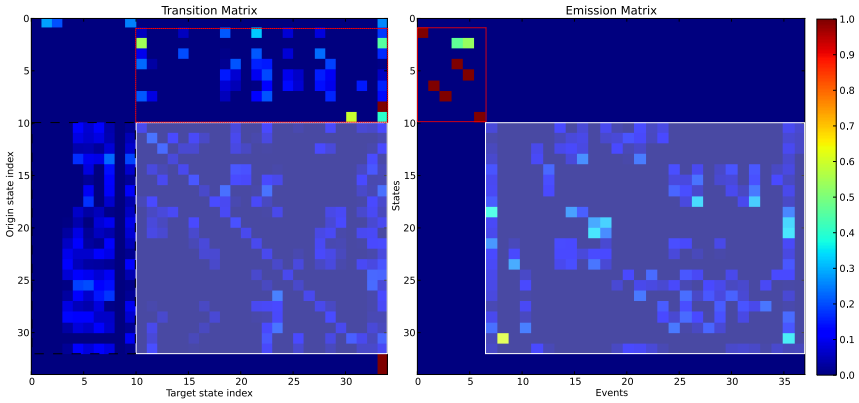


Figure 6: An illustration of a cinematographic style represented with a transition and an emission matrices (figure detailed in section 4.3).

5.1. Learning data

Regarding the selection of datasets for the learning process, the sequences we used are only taken from dialog scenes (to conform with the narrative values of dialog events we rely on). These scenes are first annotated to obtain two sequences: (i) a sequence of events and (ii) the corresponding sequence of shots. Shot annotations are performed manually due to the difficulty of automatically deciding the type of shots or the actors involved (a challenging problem with characters not facing the camera). While tedious, the task is relatively easy. In comparison, the annotation of events poses the problem of selecting an appropriate narrative value associated with every dialog event. The sequences are then automatically analyzed to insert meta-events and prep-shots. Note that the resulting event sequence and the shot sequence act respectively as input and output of the learning process described by Algorithm 1.

5.2. Maximum likelihood (ML) computation

Let $D = \{d^{\{k\}}, k \in [1,|D|]\}$ be the learning dataset composed of $|D|$ sequences, assumed to be independent, of hidden and observed variables. This hypothesis yields:

$$\log p(D|\Theta) = \log \prod_k p(d^{\{k\}}|\Theta) = \sum_k \log p(d^{\{k\}}|\Theta)$$

If the set of training data D is complete (contains both hidden and observed variables of the HMM), each data sequence $d^{\{k\}}$ is represented as a couple $(s^{\{k\}}, e^{\{k\}})$ where $s^{\{k\}}$ and $e^{\{k\}}$ respectively denote the annotated sequences of shots and events . We then have:

$$\begin{aligned}
\log p(D|\Theta) &= \sum_k \log[p(s_1^{\{k\}}|\Theta) p(e_1^{\{k\}}|s_1^{\{k\}}, \Theta) \\
&\quad \times \prod_{i=2}^{|d^{\{k\}}|} p(e_i^{\{k\}}|s_i^{\{k\}}, \Theta) p(s_i^{\{k\}}|s_{i-1}^{\{k\}}, \Theta)] \\
&= \sum_k \log p(s_1^{\{k\}}|\Pi) + \sum_i \sum_k \log p(s_i^{\{k\}}|s_{i-1}^{\{k\}}, \mathcal{A}) \\
&\quad + \sum_i \sum_k p(e_i^{\{k\}}|s_i^{\{k\}}, \mathcal{B}) \qquad (9)
\end{aligned}$$

The log likelihood is decomposed into local and independent terms of likelihood relating each variable to at most one other variable. The ML estimation can therefore be simplified by computing A, B, and Π in equation 8 as normalized contingency tables of variables e and s.

$$a_{ij} = \frac{\sum_k \sum_{l=1}^{k-1} (s_l^{\{k\}} = x_i)(s_{l+1}^{\{k\}} = x_j)}{\sum_k \sum_{l=1}^{k-1} (s_l^{\{k\}} = x_i)} \qquad (10a)$$

$$b_{ij} = \frac{\sum_k \sum_{l=1}^{k} (s_l^{\{k\}} = x_i)(e_l^{\{k\}} = o_j)}{\sum_k \sum_{l=1}^{k} (s_l^{\{k\}} = x_i)} \qquad (10b)$$

$$\pi_i = \frac{\sum_k (s_1^{\{k\}} = x_i)}{k} \qquad (10c)$$

Note that a_{ij}, b_{ij} and π_i can easily be updated when new data is available if we save the count of already used data.

Nevertheless, this solution is statistically significant if, and only if, the set D fulfills two conditions:

- D covers all the possible events in the set O;
- there is enough realizations of each shot $s = x$ and each event $e = o$ in D so that the statistics in the contingency matrices make sense.

For a single HMM, with a fully connected n-state model there are n^2 transition probabilities to learn. Therefore, considering the 34 states in our representation, we need to learn

1156 parameters in the transition matrix. By including the parameters of the emission matrix we will need to learn a total number of around $5K$ parameters. The size of the training data necessary to reach a statistically reliable result is hence very large. Furthermore, there may be some disparity between the amount of data in the dataset available for each variable.

A solution could consist in relying on existing databases of annotated movies such as the Cinemetrics repository (www.cinemetrics.lv) or Barry Salt's repository (www.cinemetrics.lv/satltdb.php).

Unfortunately such large databases lack precision and standards in the tags used for annotation, mainly due to the fact that the annotation work is performed through open contributions. Furthermore, the annotations are only related to the types of shots and do not refer to occurring actions or events in the scene, actors in shots or compositions of shots.

A closer look at our model in Figure 6 actually shows that it is not fully connected. Indeed, an HMM-director is a sparse graph and non zero parameters are rare. Furthermore, in practice, directors generally rely on a small subset of shots in their editing process as pointed out by [Sal03]. In a practical way, both considerations reduce the number of parameters to learn.

In [KGMS12], the authors consider that HMMs learned from a limited training data could not model a complex application. However, incremental learning techniques based on prior models (as shown hereafter) allows the convergence of the learning process. Indeed, incremental learning refines the parameters by using a *maximum likelihood estimation*. It improves the accuracy of the learned parameters whenever new data are available. In our work, we follow this idea and use the incremental EM algorithm (other techniques could be found in [KGMS12]) for the ML estimation.

5.3. Maximum likelihood estimation

No tractable algorithm is available for solving the problem of lack of information (scarce or incomplete data). However, the local maximum likelihood can be derived efficiently and estimated using an Expectation-Maximization (EM) process, assuming a prior about the parameters of the modeled process for the generic case. For this purpose, the Baum-Welch algorithm [Rab89] or the Baldi-Chauvin algorithm are special EM algorithms adapted to HMMs.

To estimate ML, we here rely on Equation 10 and on the Baum-Welch algorithm (as described in Algorithm 1). Let us assume that the parameters $\theta_0 = \{X, O, \Pi_0, A_0, B_0\}$ correspond to the initial parameters of the HMM. The learning task consists in computing an iterative update of θ_i, while taking into account each sequence in its entirety. The HMM parameters are updated for each new scene used for leaning until the HMM model converges to the style we want to reproduce. When the sequence contains both shots and events, θ_i is adjusted directly according to Equation 10 and if only the event sequence is available (the case when using new scripts for decoding), Baum-Welch algorithm [Rab89] is used. This convergence can be more or less fast depending on the initial values of the parameters. Therefore, this initialization should: (i) ensure a fast convergence of the learning process, and (ii) guarantee that the chosen parameters actually represent a given style.

In our initialization process, the parameters A_0, B_0 and Π_0 of the initial hidden Markov model can therefore be manually fixed so as to encode academic cinematography conventions taught in film textbooks [Ari76] (see Figure 6). We first initialize all the possible transitions in the transition matrix (Equation 11a). We then initialize the same way all the possible pairs (shot, event) (Equation 11b):

$$a_{ij}^0 = \begin{cases} 1/n_i & \text{if the transition } x_i \text{ to } x_j \text{ is allowed,} \\ 0 & \text{otherwise} \end{cases} \tag{11a}$$

$$b_{ij}^0 = \begin{cases} 1/m_i & \text{if the event } y_j \text{ can be shot by } x_i, \\ 0 & \text{otherwise} \end{cases} \tag{11b}$$

$$\pi_i^0 = \begin{cases} 1/n & \text{if } x_i \text{ can be an initial shot,} \\ 0 & \text{otherwise} \end{cases} \tag{11c}$$

with :

$$\sum_i a_{ij} = \sum_i b_{ij} = \sum_i \pi_i = 1,$$

where n_i is the number of authorized transitions from state x_i, m_i the number of events that can be shot using shot x_i, and n the number of possible initial shots. This initialization is assumed to be a style-free cinematographic editing model because it represents all possibilities of transition with the same probabilities, and avoids continuity errors between shots.

Recall that, before using an HMM for a decoding process, θ, is smoothed (all zero values are replaced by a positive value that is insignificant compared with other values) to make sure that a sequence of shots can be found for every script.

6. Cinematographer

Automatically positioning a virtual camera in a 3D environment, given the specification of our shot's properties to be satisfied (on-screen layout of subjects, visibility, shot type), is a complex problem. Most approaches address the problem by expressing visual properties as constraints or functions to optimize in order to determine the camera parameters, and rely on computationally expensive search techniques to explore the solution space.

However, an efficient solution to the automatic computation of viewpoints from shot descriptions and more precisely from the specification of on-screen compositions, has

been proposed by Lino and Christie [LC12]. The authors propose a manifold surface representing all the viewpoints in the scene that satisfy the exact location of two targets using on-screen coordinates. Each viewpoint on the surface is identified by its 2D coordinates on the manifold surface.

In our approach, we therefore map a given type of shot (*e.g.* , medium over the shoulder shot over A and B, and focusing on character A) to a default screen composition (desired on-screen location of targets), and a 2D point on the manifold surface parameterized by (θ, ϕ) with a given field of view α. This representation then enables the efficient computation of a full camera configuration (position, orientation and field of view) on the scene given the 3D coordinates of the two characters A and B.

7. Shot Generation

Once the HMM parameters have been learned, the *Director* generates the optimal shot sequence $\dot{s}$, using the *decoding* process as described in Figure 4, for a sequence of events $\hat{e}$ (real events or meta-events). Decoding consists in associating a shot s_i with each event e_i so that $\dot{s} = s_1, s_2, \cdots, s_N$ is the solution of the equation 1 using the learned style represented as an HMM.

Before determining the sequence $\dot{s}$ in equation 1, let us develop this equation. We first recall the Kolmogorov definition applied to our parameters:

$$p(\hat{s}|\hat{e}) = \frac{p(\hat{s}) \cdot p(\hat{e}|\hat{s})}{p(\hat{e})} \tag{12}$$

Now, if we replace $p(\hat{s}|\hat{e})$ in the equation 1, we obtain:

$$\dot{s} = \arg\max_{\hat{s}} \frac{p(\hat{s}) \cdot p(\hat{e}|\hat{s})}{p(\hat{e})} \tag{13}$$

The denominator of equation 12 or 13 is independent from the argument to maximize. Therefore, equation (1) becomes:

$$\dot{s} = \arg\max_{\hat{s}} p(\hat{s}) \cdot p(\hat{e}|\hat{s}) \tag{14}$$

We know that there is no dependency between any shot s_i and any event e_j except if $i = j$ (see Figure 3). Furthermore, there are no dependencies between events e_i. Therefore:

$$p(\hat{e}|\hat{s}) = \prod_{i=1}^{N} p(e_i|s_i)$$

On the other hand, the Markov assumption on the sequentiality of shots gives:

$$p(\hat{s}) = p(s_1) \prod_{i=2}^{N} p(s_i|s_{i-1})$$

Now, we replace these values of $p(\hat{s})$ and $p(\hat{e}|\hat{s})$ by their respective expressions in equation 14. Then we can deduce that:

$$\dot{s} = \arg\max_{\hat{s}=s_1 \cdots s_N} p(s_1)p(e_1|s_1) \prod_{i=2}^{N} p(s_i|s_{i-1})p(e_i|s_i) \tag{15}$$

To compute $\dot{s}$ in Equation 15, we can use the well-known Viterbi algorithm which is exactly adapted to this problem [FJ73]. The Viterbi algorithm is very efficient and its complexity is equal to $O(N \times |X|^2)$. It ensures, given the sequence $\hat{e}$, the provision of the best sequence of shots $\dot{s}$ according to the learned style.

7.1. Viterbi Algorithm

The Viterbi algorithm was introduced by Andrew Viterbi in [FJ73] as a dynamic programming algorithm which computes the optimal sequence without using an exhaustive search process. It determines, from an observation sequence $\hat{e}$, the most likely sequence of hidden states $\dot{s}$ (also called: *Viterbi path*), that might generate it.

To compute $\dot{s} = s_1, s_2, \cdots, s_N$ using the Viterbi algorithm, let:

- $\hat{e}_{u:v}$ denote a sub-sequence of events from $\hat{e}$ beginning from e_u and finishing at e_v included.
- $\tilde{s}_t(x_i)$ denote the best shot sub-sequence of length t and finishing by the shot $s_t = x_i$.

$$\tilde{s}_t(x_i) = \arg\max_{\hat{s}=s_1 \cdots s_t} p(\hat{s}|\hat{e}_{1:t-1}) \cdot p(s_t = x_i|\hat{s}_{t-1}) \tag{16}$$

- $\sigma_t(x_i)$ denote the best partial probability of reaching the intermediate state x_i at the time t.

$$\sigma_t(x_i) = p(\tilde{s}_t(x_i)|\hat{e}_{1:t}) \tag{17}$$

From the definitions below, we can easily see that:

$$\dot{s} = \arg\max_{k} \tilde{s}_N(x_k) \tag{18}$$

Calculating sub-sequences

σ can be determined through a recursive computation. To this end, we compute partial probabilities σ as the most probable route to our current position given a known HMM.

When $t = 1$ the most probable path to a state does not exist. We however use the probability of being in that state given $t = 1$ and the observable event e_1 of $\hat{e}$.

$$\forall k, \quad \tilde{s}_1(x_k) = (x_k) \tag{19a}$$

$$\sigma_1(x_k) = \pi_k \times b_{k,e_1} \tag{19b}$$

where $b_{ij} = b_{i,y_j} = p(y_j|x_i)$.

Now, we show that the partial probabilities σ_t at time t can be calculated in terms of the vector σ_{t-1} which denotes the σ's at time $t-1$. This yield:

$$\tilde{s}_t(x_k) = (\tilde{s}_{t-1}(x_j), x_k) \tag{20a}$$

$$\sigma_t(x_k) = \max_i (\sigma_{t-1}(x_i) \times a_{ik} b_{k,e_t}) \tag{20b}$$

where :

$$j = \arg\max_i(\sigma_{t-1}(x_i) \times a_{ik})$$

7.2. Limitations in Viterbi algorithm and advanced decoding

Using the Viterbi algorithm to decode an observation sequence reduces the computational complexity by using a recursive computation. However, the process is not designed in mind to integrate extra constraints in the search. Typically, aspects such as control of the shot duration (i.e. the duration for which a same state is repeated in an HMM) are not considered.

Shot duration

To handle the shot duration, we propose to introduce a small modification to the Viterbi algorithm by adding an additional cost for the computation of the σ's. Let $d(e_i)$ denote the duration of the event e_i and T_{max} the maximum duration accepted for a shot. First, we suppose that the semantic segmentation of the script produces events with a duration $d \leq T_{max}$.

$\ell_t(x_k)$ stands for the number of successive repetitions of the shot x_k in a subsequence $\tilde{s}_t(x_k)$. To handle the problem of duration we need to update the computation of sigma as follows:

$$\tilde{s}_t(x_k) = (\tilde{s}_{t-1}(x_j), x_k) \tag{21a}$$

$$\ell_t(x_k) = \begin{cases} \ell_{t-1}(x_j) + 1 & \text{if} \quad j = k \\ 1 & \text{otherwise} \end{cases} \tag{21b}$$

$$\sigma_t(x_k) = \begin{cases} 0 \quad \text{if} \quad \ell_t(x_k) \times d(x_k) \geq T_{max} \\ \max_i(\sigma_{t-1}(x_i) \times a_{ik} b_{k,e_t}) \quad \text{otherwise} \end{cases} \tag{21c}$$

with

$$j = \arg\max_i(\sigma_{t-1}(x_i) \times a_{ik})$$

Frequency of occurrence of shots

In particular scenes, it is sometimes necessary to show a specific object at least once. Shots that portray these objects are here referred to as symbolic shots. However, a straightforward application of our HMM Director model cannot integrate such a constraint. In this section, our purpose is therefore to modify the decoding algorithm to force such symbolic shots, at the best time in the sequence and with a minimum cost on the trained style.

The main idea, to control the occurrence of the symbolic shot x_r, is to define two extensions of the decoding algorithm: the "*AtMostOne* shot" and the "*AtLeastOne* shot". The *AtMostOne* constraint can be handled by making a temporary update on the trained HMM for every subsequence $\tilde{s}(x_r)$. The update consists in removing all future input transitions to x_r by forcing all a_{ir} to 0 value and adding a new variable to Equation 21 to indicate whether the original HMM or the modified HMM should be used. This update penalizes all future consideration of the shot x_r.

The "*AtLeastOne*'s constraint" requires a more complex process. We proceed with several (N) different computations of $\tilde{s}$ by forcing each computation to integrate the shot x_r on Equation 21. This task is performed by modifying the j-th computation of Equation 21. For the n^{th} calculation, we obtain:

$$j = \begin{cases} \arg\max_i(\sigma_{t-1}(x_i) \times a_{ik}) & \text{if} \quad n \neq k, \\ r & \text{otherwise} \end{cases}$$

The two constraints can be used at the same time to ensure that a particular shot appears only once.

8. Results:

In this section we show some results demonstrating the stability of the learning process and the efficiency of our method regarding the learning of styles from real data. These styles are then successfully applied to a computer generated movie. We also show the capability of our approach to add some cinematographic constraints to the learned styles.

8.1. Validation of the learning process

Given a TV series (Breaking Bad, season 3), we annotated 29 dialog scenes, directed by Michele Mclaren, from which we learn the director style as presented in section 5. Data necessary for the learning stage, learned HMM parameters, and test scene descriptions are available in the following repository: `cinematography.inria.fr`.

The validation of the learned parameters can not be performed using classical techniques (cross-validation, bootstrapping, ...) because it is hard to define a metric of similarity between two movie sequences in terms of cinematographic style.

Nevertheless, data over-fitting and model stability are verified as follows:

1. construct HMM h by computing its parameters θ from the learning dataset $D(E,S)$ using equation 6, where E represents the set of event sequences and S the set of corresponding shot sequences.
2. associate with the events set E the computed editings S' corresponding to HMM h by using equation 18.
3. construct a new HMM h' by computing its parameters θ' from the learning dataset $D'(E,S')$ by using equation 6
4. compare hmm h and h' through their parameters θ and θ'.

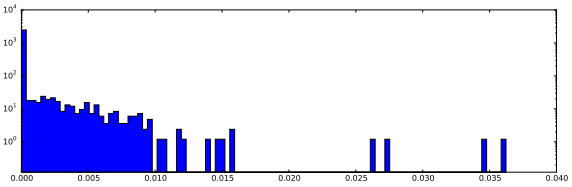


Figure 7: Histogram: bins (horizontal axis) are equal to the absolute distances between two HMM parameter sets θ (learned from original scenes) and θ' (learned from the computed scenes).

Using our data from the Breaking bad series, we compare in Figure 7 HMM h (learned from original scenes) with HMM h' (learned from the computed scenes). To this end, we compute the absolute difference between each pair of corresponding parameters within θ and θ'. Then we compute a histogram, the bins of which are the obtained difference values as shown in Figure 7.

8.2. Extracting and applying styles

Applying a learned style to new events

Here, we apply the learned style to two scene events. Both scenes are chosen from from "Breaking-bad" series. The first is the *"Contract"* scene from "Shotgun" episode. This scene is directed by M. Mclaren, as well as all those used as learning data. The second is the *"Yeah, Science !"* from "A No-Rough-Stuff-Type Deal" episode which is directed by another director.

For the *"Contract"* scene (top row in Figure 8), the sequence obtained with our method is similar to the real one, which demonstrates the capability of our system to reproduce some aspects of the editing process (viewpoints, and therefore transitions are mostly the same). Only four differences are observed. First, at the 7^{th} second, "Walter" appears in the original edition while it is does not in the computed sequence. This can be explained by the fact that Walter has not yet been assigned a discourse role in the scene. In fact, as we claimed, we are taking into account only the discourse events. And since Walter is not participating in the discourse at this time it is expected that our technique will not show him (except in overall shots). Then in interval [25s, 33s], our approach produces a medium-over-the-shoulder shot, while in the original movie a medium shot is used. This can be explained by the fact that the computed sequence keeps on reproducing the general style whereas the director proceeds differently to express some particular feeling. The same observation can be made for the interval [11s, 14s]. Finally, at the end of the sequence, our method produces a full-body-over-the-shoulder shot while an overall shot is used in the real movie. This is due to the fact that the last shot in a scene depends on the next scene which is not considered in the experiment shown in Figure 8. this observed variation, between the last original and the last computed shots, is therefore not significant.

To sum up, in this sample 78% of the generated shots in the computed sequence are identical to those in the original sequence. In addition, actor compositions of the shots are identically reproduced in 74% of the generated shots, and partially reproduces (same focused actor) in 9%.

8.3. First-order Markov Model vs HMM

We compare our HMM-based approach to a classical first-order Markov model (1MM) on the same dataset. In a 1MM representation, each type of shot is encoded as a state, and probabilities of transitions between shots are specified by a transition matrix. 1MM representations make a decision to select a shot for each new event. In other words, it chooses the next shot depending only the current shot and the next event (local decision). In comparison, our method selects the optimal sequence of shots while considering all the sequence of events. The necessity to objectively compare the two methods requires some changes. Indeed, for 1MM, we propose to take into account the whole sequence of events using the Viterbi algorithm and use the same learning dataset as the one used for HMM. The resulting shots are compared to the original editing as shown in figure 9.

We can notice that 1MM fails to reproduce the learned style whereas our approach follows it faithfully. This supports the hypothesis that the editing process is non-Markovian.

8.4. Actor hierarchy

In this section, we show that our technique can produce high editing variations corresponding to the variation on actor importance. Let us take the famous *"Yeah, Science"* scene from Breaking-bad series. In this scene, Jesse is the important actor since the director wanted to show his reaction to a previous event. Figure 10 shows that for the same HMM, when we inverse the hierarchy (here put Walter as principal actor) the result sequence is totally different.

8.5. Pacing constraint

We want to show that our method is capable of changing the pace of certain shots while applying a given style. For this purpose, we consider a simple 3-state HMM (over-the-shoulder focusing on the principal actor, over-the-shoulder focusing on secondary actor and symbolic shot). The top row in figure 11, represents 2 computed sequences: the green sequence corresponds to the considered HMM without constraints, whereas the red sequence fulfills a pace constraint

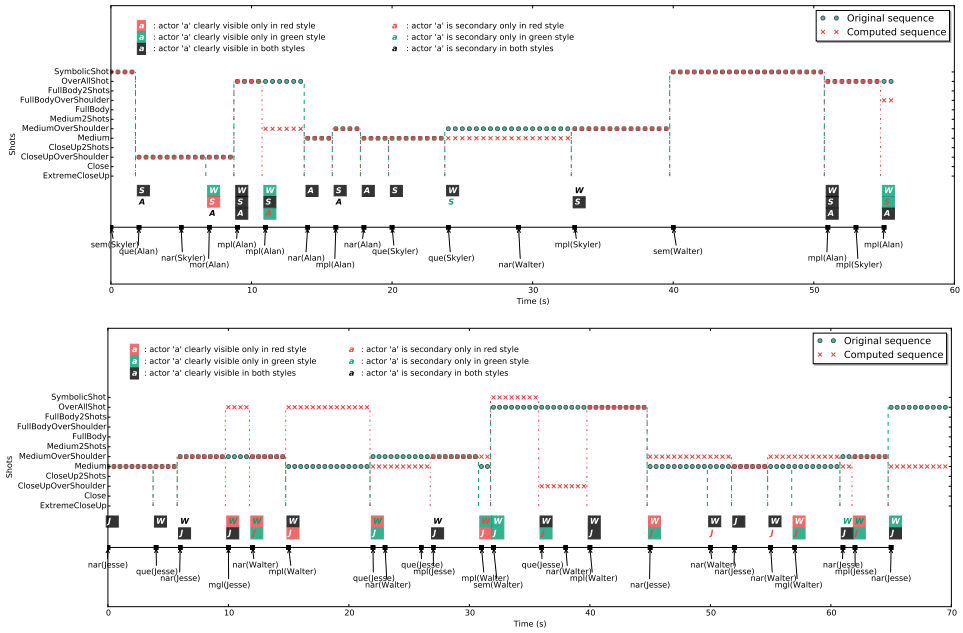


Figure 8: Learned HMM (from MMacLaren Style) applied: (top row) to the events of the *"Contract"* scene; (bottom row) to the events of the *"Yeah, science !"* scene. The generated shot sequence is similar to the original in the top row scene which is edited by the same director for which we learned the style, while it is different in the bottom row scene edited by another director. Vertical axis: shot types.

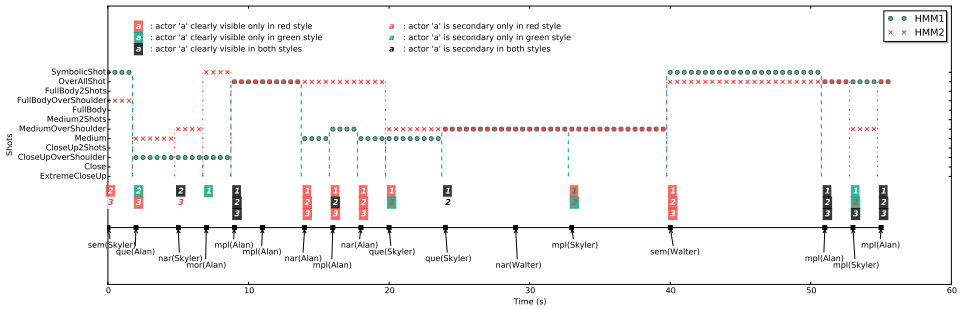


Figure 9: 1MM aplyed to "Contract" actions (refer to numerical format).

which consists in limiting the duration of the shots to 7 seconds. We can observe that this constraint is fulfilled for all the shots.

8.6. Inserting a symbolic shot

Given a directing style, our objective is to constrain the use of a new shot (*e.g.* symbolic shot) in the generated sequence of shots so that it appears at least once in the sequence. Middle and bottom rows in Figure 11 illustrate how this constraint is considered. We observe that the decoding algorithm modifies one or more shots located in the close neighbourhood of the inserted shot (see bottom row sub-figure).

9. Conclusion and Future work

We proposed a virtual director based on a hidden Markov model in order to tackle the problem of shooting a virtual scene with cinematographic styles either learned from real movies or manually edited by a user. To this end, we modeled the problem as an HMM in which the hidden states represent the shots while the observations correspond to the script events of the scene to shoot. Determining a sequence of shots (hidden states) from the input sequence of the events (observations) is performed through a decoding operation. We have shown that after learning a given style from real movies, our method is able to efficiently reproduce it. We

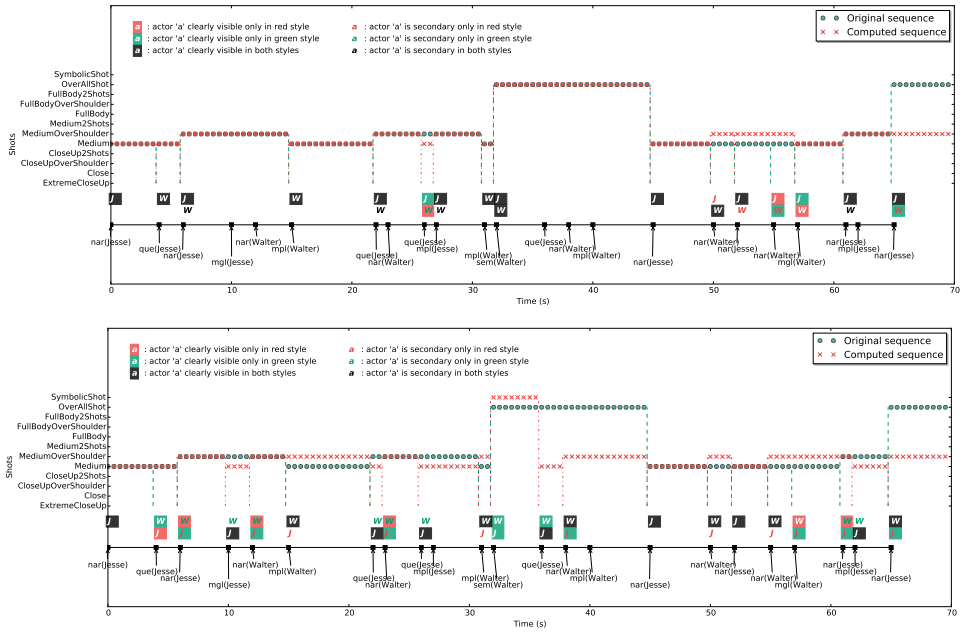


Figure 10: Same HMM-director applied to the "Yeah, Science !" actions, (top) with Jesse as principal actor, and (bottom) Walter as principal actor.

have also demonstrated that our HMM-based approach outperforms finite state machine representations in terms of style reproduction. Second, we have improved the decoding algorithm of HMMs to consider additional cinematographic constraints such as pacing or insertion of symbolic shots in the scene while respecting the learned style. Third, unlike other approaches, our method does not need any prior knowledge of the scene geometry and staging.

As future work, we propose to study how to combine different learnt styles and perform transitions between them, as well as proposing more evolved representations to handle discourse structures such as parallel stories, flashbacks and foreshadows. Besides its application to automated cinematography, this work may also be used as a way to help analyzing styles and discourse of real movies.

References

[Ari76] ARIJON D.: *Grammar of the Film Language*. Hastings House Publishers, 1976. 2, 9

[AWCO10] ASSA J., WOLF L., COHEN-OR D.: The Virtual Director: a Correlation-Based Online Viewing of Human Motion. *Computer Graphics Forum 29*, 2 (January 2010), 595–604. 2

[BLBA08] BOUDAREN M. E. Y., LABED A., BOULFEKHAR A. A., AMARA Y.: Hidden markov model based classification of natural objects in aerial pictures. In *Proceeding of IAENG International Conference on Signal and Image Engineering, London* (2008). 4

[Bou13] BOUSSION T.: Construire un dialogue hollywoodien (apocalypse now, 300, eyes wide shut...). http://analyse-scenarios.over-blog.com/, July 2013. Internet Blog. 6

[BPSW70] BAUM L. E., PETRIE T., SOULES G., WEISS N.: A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics* (1970), 164–171. 5

[CAH*96] CHRISTIANSON D. B., ANDERSON S. E., HE L.-W., SALESIN D. H., WELD D. S., COHEN M. F.: Declarative camera Control for Automatic Cinematography. In *AAAI'96 Proceedings of the Thirteenth National Conference on Artificial Intelligencec* (1996), vol. 1, pp. 148–155. 1, 2, 4

[ER07] ELSON D. K., RIEDL M. O.: A Lightweight Intelligent Virtual Cinematography System for Machinima Production. In *3rd Annual Conference on Artificial Intelligence and Interactive Digital Entertainment* (2007), pp. 8–13. 1, 2, 3, 5

[FJ73] FORNEY JR G. D.: The viterbi algorithm. *Proceedings of the IEEE 61*, 3 (1973), 268–278. 10

[GRLC14] GALVANE Q., R. R., LINO C., CHRISTIE M.: Continuity Editing for 3D Animation. In *Processdings of AAAI Conference on Artificial Intelligence* (2014). 5, 6

[HCS96] HE L. W., COHEN M. F., SALESIN D. H.: The Virtual Cinematographer: A Paradigm for Automatic Real-time Camera Control and Directingh. In *SIGGRAPH '96 Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (1996), pp. 217–224. 1, 2, 3, 5

[JY05] JHALA A., YOUNG R. M.: A Discourse Planning Approach to Cinematic Camera Control for Narratives in Virtual Environments. In *AAAI'05 Proceedings of the 20th National Conference on Artificial Intelligence* (2005), vol. 1, pp. 307–312. 1, 2

[KC08] KARDAN K., CASANOVA H.: Virtual cinematography of group scenes using hierarchical lines of actions. In *Proceedings*

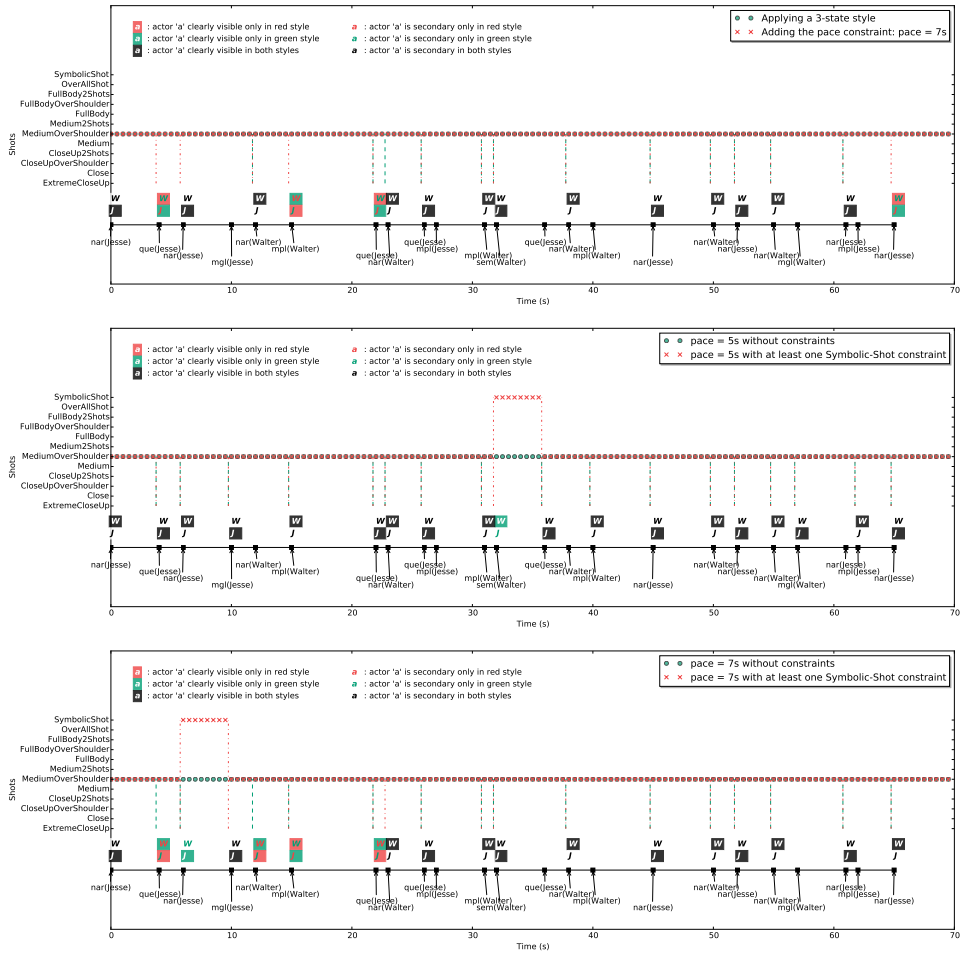


Figure 11: (Top) Basic 3-state HMM with and without pacing constraint on POC events. (Middle) Basic 3-state HMM with and without the "at least one" constraint with a 5 seconds pacing. (Bottom) Basic 3-state HMM with and without the "at least one" constraint with a 7 seconds pacing constraint.

of the 2008 ACM SIGGRAPH Symposium on Video Games (New York, NY, USA, 2008), Sandbox '08, ACM, pp. 171–178. 6

[KGMS12] KHREICH W., GRANGER E., MIRI A., SABOURIN R.: A survey of techniques for incremental learning of hmm parameters. *Information Sciences 197* (2012), 105–130. 9

[LC12] LINO C., CHRISTIE M.: Efficient composition for virtual camera control. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2012), Eurographics Association, pp. 65–70. 5, 6, 10

[LCCR11] LINO C., CHOLLET M., CHRISTIE M., RONFARD R.: Computational Model of Film Editing for Interactive Storytelling. In *ICIDS 2011 - International Conference on Interactive Digital Storytelling* (Vancouver, Canada, Nov. 2011), Si M., Thue D., André E., Lester J. C., Tanenbaum J., Zammitto V., (Eds.), vol. 7069 of *Lecture Notes in Computer Science*, Springer, pp. 305–308. 2, 3

[LCL*10] LINO C., CHRISTIE M., LAMARCHE F., SCHOFIELD G., OLIVIER P.: A real-time cinematography system for interactive 3d environments. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2010), Eurographics Association, pp. 139–148. 2, 3, 5

[LCRB11] LINO C., CHRISTIE M., RANON R., BARES W.: The director's lens: An intelligent assistant for virtual cinematography. In *Proceedings of the 19th ACM International Conference on Multimedia* (New York, NY, USA, 2011), MM '11, ACM, pp. 323–332. 3

[Rab89] RABINER L.: A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE 77*, 2 (1989), 257–286. 4, 5, 9

[Sal03] SALT B.: *Film Style and Technology: History and Analysis (2nd edition)*. Starword, 2003. 9

[Vit67] VITERBI A. J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on 13*, 2 (1967), 260–269. 5

[War03] WARD P.: *Picture Composition for Film and Television*. Media production. Focal Press, 2003. 6